Theater Key Retrieval (TKR)

A System for Automated KDM Delivery

1 Introduction

In Digital Cinema, the generation of KDMs is typically a highly automated process. However the delivery of those KDMs to the equipment in the field has been performed by various ad hoc delivery mechanisms. Worse, these mechanisms either rely on proprietary VPNs or are plagued with manual workflows (e.g. email with attachments, USB sticks, etc.).

This document will provide a simple standardized framework for the automated delivery of KDMs from service providers to equipment in the theaters using the Internet. In addition, TKR aims to be both secure and easy to implement for device vendors as well as service providers.

2 Overview

So far, most of the proposed network-based delivery mechanisms require that a theater enable a server that listens for incoming connections from the Internet. In such an approach, each theater would be burdened with such tasks as: obtaining static IP addresses (or coping with the challenges of dynamic ones), managing DNS records, configuring proper firewalls to permit inbound connections, and maintaining the specifics in the TDL.

TKR attempts to resolve these issues by choosing that devices *in the theater* initiate the network connection. Doing so eliminates all of the above IT requirements at the theater.

In TKR, KDMs are simply published on a website, and automatically retrieved by devices in the field. These devices will, in essence, identify URLs from within their CPLs and use these URLs to retrieve their KDMs.

3 Requirements

The requirements associated with TKR are split across two entities: the *Content Author* and the *In-Theater Device*.

3.1 Content Requirements

TKR enabled content shall be identified by the content's Composition Playlist. The CPL's `Issuer` element shall contain the XML attribute: `language="x-TKR"`. The text value of the `Issuer` element shall be a URL whose scheme is either "`http`" or "`https`". This URL, when concatenated with the hex encoded Public Key Thumbprint of a Security Manager's leaf certificate, shall resolve to a KDM Bundle containing all authorized KDMs for that combination of device and CPL.

Note: The URL embedded in the CPL is termed the "Base URL", while the final URL formed by concatenating the thumbprint is termed the "Bundle URL".

The Base URL shall be unique to the CPL in which it is contained. This uniqueness may be achieved using that CPL's own `Id` value.

3.1.1 Public Key Thumbprint

The Public Key Thumbprint used to form the Bundle URL shall be computed as per Section 5.4 of SMPTE 430-2. In order to be URL safe (i.e. to not require special URL encoding), the value shall be represented in base-16 (i.e. hex) using the symbols `0-9` and the lower-case symbols `a-f`.

Note that per SMPTE 430-2, the dnQualifer X.509 Name Attribute will contain the Public Key Thumbprint in base64 format. One may simply convert the dnQualifier value from base64 to base-16 in order to form the Bundle URL.

3.1.2 Unencrypted Content

Unencrypted content (i.e. content not requiring a KDM) shall not be mastered with a TKR Base URL.

3.2 Device Requirements

In the theater, TKR capable devices may include Screen Management Systems, Theater Management Systems or Security Managers. Regardless of the type, the requirements are identical. The following section outlines the requirements for in-theater devices that support TKR.

3.2.1 Internet Connectivity

TKR capable devices shall have outbound access to the Internet, and should be configured to resolve hostnames with DNS. Outbound TCP connections on port 80 and 443 shall not be restricted. There is no requirement for incoming connections from the Internet.

Implementers may provide their operators with a simple means to validate that the above requirements have been met. This could be achieved by performing an HTTP GET to a known fixed URL. A successfully HTTP Response of 200 would indicate that both DNS and routing are correctly configured.

3.2.2 KDM Retrieval

A TKR device shall periodically perform the following steps on each piece of *encrypted* content to which it has access[1]:

1) Examine the `Issuer` element of the CPL. If its `language` attribute is "`x-TKR`" then proceed.
2) Extract the text value of the `Issuer` element which shall be the Base URL.
3) Form the Bundle URL by appending the hex encoded Public Key Thumbprint of the Security Manager's leaf certificate to the Base URL.
4) Perform an HTTP GET on the Bundle URL[2].
5) The response shall be a KDM Bundle (served with the HTTP Header "`Content-Type: application/tar`").
6) Unpack the KDM Bundle and ingest any of the resultant KDMs that have not yet been ingested.

If no KDMs are authorized at that moment, the device should expect the server to respond with the HTTP Status "`404 Not Found`". Such a response shall imply nothing about the future availability of KDMs.

3.2.3 KDM Ingest Receipt

TKR enabled devices should acknowledge the successful ingest of all *new* KDMs they receive. This acknowledgment shall be made by submitting an HTTP POST request to the corresponding CPL's Base URL once for each new KDM ingested. The content of the request shall be the `MessageId` of the ingested KDM.

If the server responds with the HTTP Status "`405 Method Not Allowed`" then the device should not retry. Such a response is an indication that that TKR host does not require Ingest Receipts. On all other errors, the device should retry the POST each day until either success or the KDM expires.

The acknowledgment of a KDM's Ingest shall not prevent that KDM from being made available to subsequent KDM Bundle requests. This policy eliminates the possibility of a kind of Denial of Service attack in which an entity other than the target device can claim receipt of the KDMs and thereby prevent receipt by the intended device.

Within the context of KDM Ingest Receipt, TKR devices should defer to the target Security Manager for its best assessment of what constitutes successful KDM ingestion.

[1] If the TKR device is acting on behalf of more than one Security Manager (e.g. a TMS), then it should in turn, perform these steps for each Security Manager to which is has access.
[2] Efficient implementations may perform multiple KDM Bundle requests asynchronously so as to preclude any one host from blocking timely access to others.

3.2.4 Automated Polling Interval

TKR capable devices should automatically perform the steps outlined in Section 3.2.2 once every 20 minutes per applicable CPL. This rate ensures a balance between the operational needs of the exhibitor and the desire to minimize load on the KDM host. This interval should be extended to 6 hours for any CPL currently unlocked with an active KDM that does not expire in the next 24 hours.

Implementers should not choose a polling algorithm that would cause all implementations in the field to poll simultaneously, such as a particular minute of the hour. Instead they should use a mechanism such as a random or fixed interval since the last update. Fixed intervals are less useful because they may still result in clumping within an administrative zone. A fixed interval randomly selected at runtime would be better.

Devices may optionally inhibit polling during content playback or other critical operations if there is a concern that such polling would adversely affect the server's operation.

3.2.5 Manual Polling

The Graphical User Interface (GUI) of a device supporting TKR should provide the means for an operator to manually initiate a request for KDMs with no regard to the above Polling Interval.

3.2.6 HTTP Feature Support

This section outlines various features of HTTP and how they should be applied to TKR.

3.2.6.1 HTTP Redirects

TKR capable client devices shall follow HTTP redirects (HTTP status codes 3xx) as described in RFC2616.

In the case of POST requests, some user agents mistakenly convert the request method from POST to GET when following a redirect. TKR clients shall respect RFC2616 and not alter the request method when following a redirect.

3.2.6.2 Basic Access Authentication

TKR capable client devices shall support Basic Access Authentication including the ability to extract credentials from URLs.

Service Providers should favor obfuscated URLs over the use of login credentials, since there is no security advantage to one over the either. In either case access to the CPL grants access to the KDMs.

3.2.6.3 ETags

Both TKR servers and clients should employ the use of HTTP ETags in order to promote efficient caching.

3.2.6.4 SSL Certificates

TKR hosts that wish to use HTTPS should avoid the use of self-signed SSL certificates and should ensure that their SSL certificate's Common Name agrees with their hostname.

To ensure robustness TKR enabled devices should not validate the server's SSL certificate against a set of trusted Certificate Authorities. Additionally clients should not enforce agreement between the server's hostname and the certificate's Common Name.

4 Appendix (Informative)

4.1 CompositionPlaylist Sample (SMPTE)

The following CompositionPlaylist is a valid instance of a TKR enabled SMPTE CPL.

```
<?xml version="1.0" encoding="UTF-8"?>
<CompositionPlaylist xmlns="http://www.smpte-ra.org/schemas/429-7/2006/CPL">
  <Id>urn:uuid:bddf9ba8-bb98-4bdc-97b0-4e4e0cf13d13</Id>
  <IssueDate>2011-11-01T18:41:09-07:00</IssueDate>
  <Issuer language="x-TKR">http://example.org/bddf9ba8-bb98-4bdc-97b0-4e4e0cf13d13/</Issuer>
  <Creator>Studio A</Creator>
  <ContentTitleText>The Jazz Singer</ContentTitleText>
  <ContentKind>feature</ContentKind>
...
```

4.2 CompositionPlaylist Sample (Interop)

The following CompositionPlaylist is a valid instance of a TKR enabled Interop CPL. Note how the Issuer element's XML attribute differs from that of the SMPTE CPL in Section 4.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<CompositionPlaylist xmlns="http://www.digicine.com/PROTO-ASDCP-CPL-20040511#">
  <Id>urn:uuid:d6469a02-cea0-4e65-afd3-bea6cbe0639d</Id>
  <IssueDate>2011-11-17T08:39:22-08:00</IssueDate>
  <Issuer xml:lang="x-TKR">http://example.org/bddf9ba8-bb98-4bdc-97b0-4e4e0cf13d13/</Issuer>
  <Creator>Studio A</Creator>
  <ContentTitleText>The Jazz Singler</ContentTitleText>
  <ContentKind>feature</ContentKind>
...
```

4.3 Base URL Samples

The following are example Base URLs that could be found within a CompositionPlaylist having an `Id` of `bddf9ba8-bb98-4bdc-97b0-4e4e0cf13d13`. They highlight the different approaches that a service provider might take in designing both their URLs and server-side architecture:

```
http://example.org/kdm-download.php?cpl-id=bddf9ba8-bb98-4bdc-97b0-4e4e0cf13d13&dev-thumbprint=

https://bob:secret@provider.com/bddf9ba8-bb98-4bdc-97b0-4e4e0cf13d13/

https://kdms.studio.com/5gj30d/bddf9ba8-bb98-4bdc-97b0-4e4e0cf13d13/
```

Each Base URL above is designed to have appended any Certificate Thumbprint to form the Bundle URL. For example in the case of the last URL above, a server with a certificate whose `dnQualifier` is "`HJgYRs2pYrt6NI390gPtdfZhs9g=`" would append "`1c981846cda962bb7a348dfdd203ed75f661b3d8`", resulting in the following Bundle URL:

```
https://kdms.studio.com/5gj30d/bddf9ba8-bb98-4bdc-97b0-4e4e0cf13d13/1c981846cda962bb7a348dfdd203ed75f661b3d8
```

4.4 Working With Public Key Thumbprint

4.4.1 Command Line

The following command line will extract the hex encoded Public Key Thumbprint from a PEM file, in this case named "sample.pem". It requires the utilities `openssl` and `dd`. Both are commonly available on POSIX platforms.

```
$ openssl x509 -pubkey -noout -in sample.pem | openssl base64 -d \
   | dd bs=1 skip=24 2>/dev/null | openssl sha1 -hex
```

4.4.2 Source Code – pubkey-thumbprint.c

The following `c` program performs the same function as the command line above. It either requires a PEM file as an argument or, with "-" as an argument, will read a PEM from stdin.

```
// -*- compile-command: "cc -o pubkey-thumbprint pubkey-thumbprint.c -lcrypto -Wall" -*-
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <openssl/sha.h>
#include <openssl/pem.h>

int main(int argc, char** argv) {

  OpenSSL_add_all_digests();

  if ( argc != 2 ) {
    fprintf(stderr, "USAGE: pubkey-thumbprint [<cert-file.pem>]\n");
    return 1;
  }

  BIO *in = BIO_new(BIO_s_file());

  if ( strcmp(argv[1], "-") == 0 )
      BIO_set_fp(in,stdin,BIO_NOCLOSE|BIO_FP_TEXT);
  else
      if (BIO_read_filename(in, argv[1]) <= 0) {
          perror(argv[1]);
          BIO_free(in);
          return 2;
      }

  X509 *x509 = PEM_read_bio_X509(in, NULL, NULL, NULL);
  if (!x509) {
      fprintf(stderr, "Error decoding file %s\n", argv[1]);
```

```
      BIO_free(in);
      return 3;
  }

  BIO_free(in);

  EVP_PKEY *key = X509_get_pubkey(x509);
  X509_PUBKEY *x_pub=NULL;
  X509_PUBKEY_set(&x_pub, key);

  uint8_t sha_value[SHA_DIGEST_LENGTH];
  SHA1(x_pub->public_key->data, x_pub->public_key->length, sha_value);

  int i;
  for (i=0; i<sizeof(sha_value); i++)
     printf("%02x", sha_value[i]);
  printf("\n");

  X509_PUBKEY_free(x_pub);
  EVP_PKEY_free(key);
  X509_free(x509);

  return 0;
}
```

5 References

[RFC1738] “Uniform Resource Locators (URL)”, 1994. IETF RFC 1738. See: http://www.ietf.org/rfc/rfc1738.txt

[RFC2616] “Hypertext Transfer Protocol – HTTP/1.1”, 1999. IETF RFC 2616. See: http://www.ietf.org/rfc/rfc2616.txt

[HTTP AUTH] “HTTP Authentication: Basic and Digest Access Authentication”, 1999. IETF RFC 2617. See http://www.ietf.org/rfc/rfc2617.txt

[HTTPS] “HTTP Over TLS”, 2000. IETF RFC 2818. See http://www.ietf.org/rfc/rfc2818.txt

[KDM] SMPTE 430-1-2006, D-Cinema Operations – Key Delivery Message

[KDMb] SMPTE 430-9-2008, D-Cinema Operations – Key Delivery Bundle

[DCERT] SMPTE 430-2-2006, D-Cinema Operations – Digital Certificate

[CPL] SMPTE 429-7-2006, D-Cinema Packaging – Composition Playlist